A Review of Session Initiation Protocol

Pritha Mahata[1] Babak Etefaghhaziri[1]

1 Dept. of Computer Systems
P.O. Box 325
S-751 05 Uppsala, Sweden
{pritha},{babake}@docs.uu.se

Abstract

The Session Initiation Protocol (SIP) is an application layer protocol for creating, modifying and terminating sessions with one or more participants.The activities in SIP can be modelled as client, server interactions. In this paper, we describe SIP architecture, SIP messages and the behaviour of clients and servers using execution sequences and state diagrams showing the order of exchanges of messages used in SIP.

Acknowledgements

We would like to acknowledge that we got important feedbacks on this document from our supervisor Parosh Abdulla, Hans Nilsson (Ericsson) and our colleague Aletta Nylen.

1 Introduction

The Session Initiation Protocol (SIP) is an application layer protocol for creating, modifying and terminating sessions (e.g, multimedia conferences, internet telephone calls) over Internet with two or more participants.

Through SIP, participants can establish multicast (one to many) connections or unicast (one to one) connections.

To establish a connection, SIP messages include information to describe the type and properties of the media and the connection to be established. To describe the type of the media, SIP uses another protocol called SDP.

SIP is only used to initiate, modify or terminate connections. The data is transferred

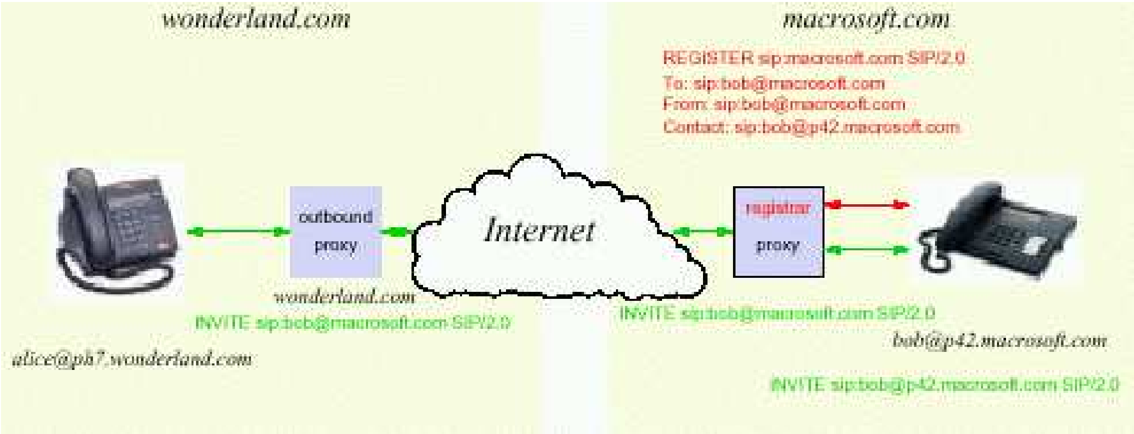

Figure 1: A SIP connection

using other protocols (e.g, TCP or UDP for Internet and X.25 for GSM networks). The data can be audio or video, although currently most of the SIP applications are used for audio transfers. (see Fig. 1)

In SIP, every participant is identified by a SIP *address* in e-mail format. The adrresses can be permanent or temporary. There is a database, called *Registrar* which binds the permanent SIP *address* to a temporary SIP address reflecting current network location. A participant updates *Registrar* about its current location, so that other servers will know at which address the participant can be reached. *Registrar* can be accessed by different participants for finding the location of each other on the Internet. There may be more than one registrar in a system and participants might need to connect to different registrars when they want to establish a session. An address of a participant can also be fetched from external databases using non-SIP protocol. A *location server* can be a registrar or an external database. To fetch an address from a *location server* for establishing a session, other protocols (e.g, LDAP - light weight directory access protocol) are used.

An permanent SIP address can be registered to more than one temporary addresses(e.g, A@xyz.com can correspond to addresses of A's both office and home computer) in a Registrar. This provides mobility to the participants in SIP. Thus, when a participant(A's friend) attempts to call another participant (A), a message is sent to all the locations registered for A to establish a connection. Once an answer is received from one of these locations, a connection is established.

Messages in SIP are grouped in two: *requests* and *responses*. *Requests* are messages sent by a participant who starts, modifies or terminates a session with other participants. *Responses* are messages that a participant recieves in return.

A SIP participant is an end-user of the protocol and to make a connection , he uses two types of components in SIP: *clients* and *servers*. *Clients* are components which generate requests and *Servers* are components which generate responses.

Outline : In Section 2, we describe the SIP architecture. Section 3 describes SIP messages. Section 4 describes execution sequences for some of the most common requests in SIP. Section 5 describes the behaviour of clients and servers using state diagrams.

2 SIP architecture as a Client-Server Distributed System:

The activities in SIP can be modelled as interactions in a distributed environment. These interactions are between two groups of components,, namely, *clients* and *servers*. Every component in SIP is either a *client* or a *server* or both (changing roles over time). A *client* in SIP is a component which sends a request to other components. A *server* in SIP is a component that generates a response according to the request it receives. SIP servers can belong to four categories depending on their role in the establishment of a connection (details can be found later in this section) . The server categories are *Proxy servers, Redirect servers, Registrars* and *User agent servers* and the *User agent client* is the sole client. *Proxy servers* and *redirect servers* contact *Location servers* by requesting a name and get an address in response. But, this is done by a non-SIP protocol.

To describe the communication with servers, we will give a short description of the messages used in the communication. The *messages* are described in Section 3. *Requests* used in the descriptions of the servers are *Invite, Ack, Bye and Cancel*. The *Invite* request is sent to invite a user to participate in a session. The Ack request (sent by the caller) confirms that the caller has received a final response from the callee in answer to the *Invite* request. The user agent client uses *Bye request* to indicate to the server that it wishes to release the call. The *Cancel* request cancels a pending request. The final response from the server is designated by *Ok* responses which mainly mean that the request was successful and the search can be terminated. The *redirect responses* means that the location has moved temporarily or permanently.

The servers and clients are described below :

1. **User agent:** *User agents* can either act as a *user agent server* when it receives a call or as a *User agent client* when it makes a call.

   - **User Agent Server:** It can respond to the requests and is an end point in a session. The callee of a session is actually a *User agent server*. When a *user agent server* receives an *Invite* request, it generates a "trying" response and notifies the user that it has received an *Invite*. If the user accepts the *Invite* request, then it generates an *Ok* response and waits to receive an *Ack* from the caller to start the media. If the user declines the Invite, the user agent server generates a response which explains whether the reason of declining was a client error, server error or global error.The *user agent server* generates an *Ok* response for a received *Cancel* or *Bye* request.

- **User Agent Client:** A *user agent client* produces requests in order to establish a session or terminate it. The caller is a *user agent client*. To establish a session, user agent client send *Invite* requests to the callee. If it receives a response indicating that more than one internet location is registered in the registrar or the internet location of the callee is temporarily or permanently changed, it generates a new *Invite* request to one of the new addresses received. If the user agent client receives a response indicating that the request was not accepted by the server, the *user agent client* sends an *Ack* indicating that it has received the response and terminates. On the other hand, after sending an Invite request, if the user agent client receives a response like "trying" or "ringing", it notifies the user that the establishment of the session is in progress and waits until it receives an *Ok* response to start the media transfer after sending an *Ack* request. User agent clients can stop the creation of a session using *Cancel* request and an ongoing session by using *Bye* request.

2. **Registrar:** A *Registrar* is a database containing the available SIP addresses for various users who can participate in a session.

3. **Proxy Server :** When a partcipant wants to communicate with some other participant, the communication may need to go via a *server*, called *Proxy server* . It can receive requests from the caller and respond to them or forward the *responses* to the caller and *requests* to the callee. The IP location of the proxy server is known to some participants. When one of these participants wants to send a request to another participant, it first sends the request to the proxy server. It is then proxy server's task to find the actual SIP address of the other participant and deliver this request. Figure 2 demonstrates an example when a proxy server is used.

   A SIP interaction through proxy server :

   A brief explanation of this figure is as following:

   A user having the SIP address **sip:pritha@docs.uu.se** is the **caller** and **babake@home** is the actual address of the callee in this senario. Caller knows **etefb@metu.edu.tr** as the SIP address of the callee and sends *Invite* to this SIP address. Proxy server requests location server for the actual address

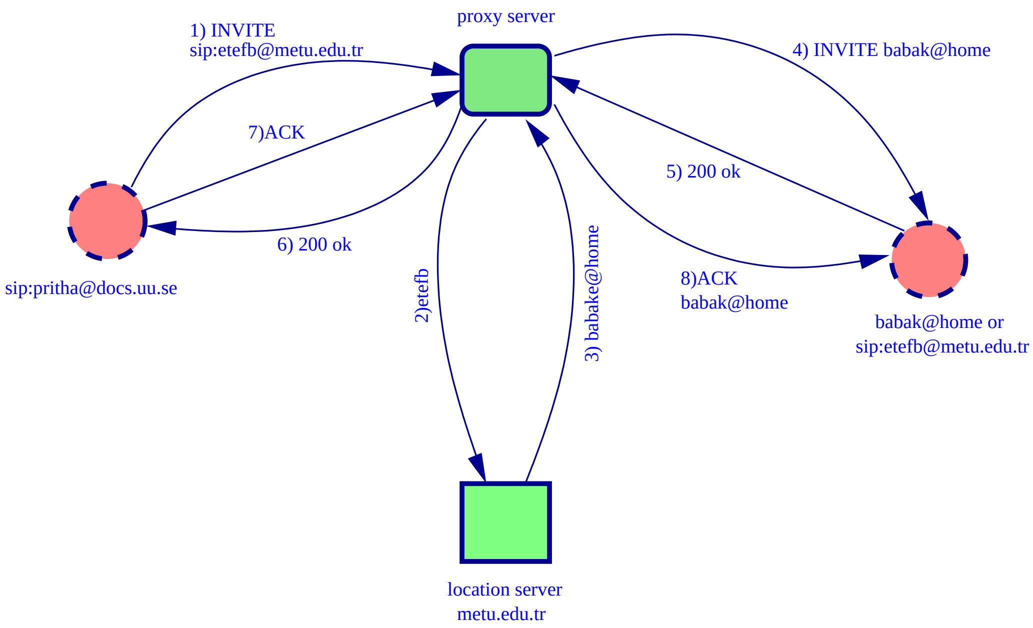

Figure 2: Connection with a proxy server

of **etefb** and receives **babake@home** as an asnwer. The communication steps are as follows :

(a) The caller sends *Invite* request to the proxy server to be delivered to the callee(see section 3 for details about requests and responses).

(b) The proxy server fetches the address of the callee from the *location server*.

(c) The *location server* sends the actual SIP address of the callee to the proxy server.

(d) The proxy server then forwards the *Invite* request to the received SIP address (to the callee).

(e) The callee is alerted.

(f) The callee sends a *response Ok* to the proxy server.

(g) The proxy server forwards this response back to the caller.

(h) The caller then generates another request *Ack* and sends it to the proxy server.

(i) This request is also forwarded to the callee by the proxy server.

Note that, during these activities, the participant who started the session was not required to know the actual IP address of the participants it communicated with. It is the proxy server which delivers the messages to different parties.

4. **Redirect Server:** A participant who wants to communicate with another participant may also need to contact a *Redirect server* to establish a connection. A *Redirect Server* finds the IP address of the callee, but it *does not forward* messages generated by a caller to the callee like a proxy server. It finds the IP address of the called participant from the *Registrar* and then responds by sending the new address to the caller. After the caller receives the IP address of the callee, it generates a new request and sends it directly to the callee. Figure 3 demonstrates an example when redirect server is used.

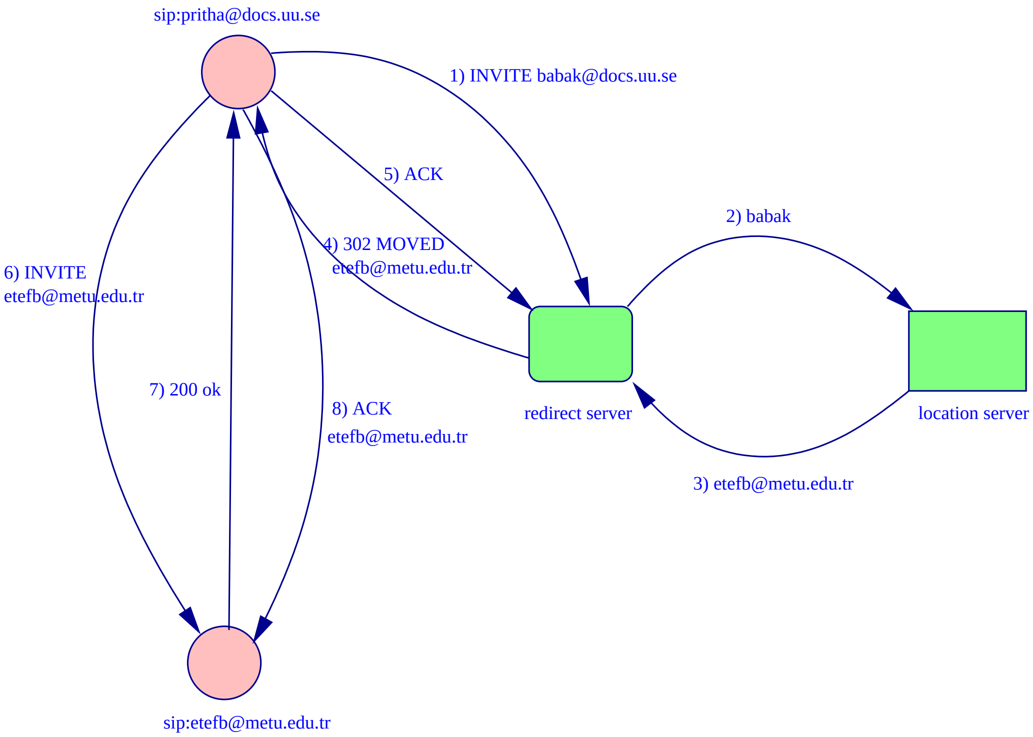

Figure 3: Connection with a redirect server

A SIP interaction using Redirect server The user identified as **pritha@docs.uu.se** is the caller and **babak@docs.uu.se** is the known address of the callee. The redirect server responds to the caller saying that the actual address of the callee is **etefb@metu.edu.tr**.

(a) Like in the previous example,, the session starts with an *Invite* request sent by the caller to the *redirect server*.

(b) The*redirect server* requests the registrar for the actual IP address of the caller.

(c) The *redirect server* receives the actual IP address of the caller from the *registrar*.

(d) The *redirect server* then responds to the caller by a response which includes the IP address of the callee.

(e) The caller then sends another request *Ack* to the *redirect server* indicating that it has received the IP address.

(f) The caller sends another *Invite* request directly to the IP address that it has received.

(g) The callee responds to the caller directly with an *Ok* response.

(h) The caller sends an *Ack* to the caller directly. Now, the connection is established.

Note: When a redirect server is used, caller directly contacts callee after it receives the IP address of the callee from the Redirect server.

In general, there might be several components involved in establishing a session. Several servers can be used along the path of communication when a message is sent from one component to the other. SIP proxy servers forward requests from the User Agent Client to the next SIP server or user agent within the network and also retain information for billing/accounting purposes. SIP Redirect Servers respond to client requests and inform them of the requested server's address. Numerous hops can take place before reaching the final destination. SIP allows both Proxy and Redirect servers to contact external location servers to determine user or routing policies, and therefore, does not bind the user into only one scheme to locate users. In addition, to maintain scalability, the SIP proxy servers can either maintain state information (statefulness, since it remebers earlier transactions) or forward requests in a stateless fashion (see Figure 4). In the latter case, no informaiton about the message is stored in the server and resending a message requires a regeneration of the message.

3 SIP messages

This section provides a classification for type of messages used in SIP. Each *message* in SIP is either a *request* or a *response*. When a SIP component receives a *request*, it returns a response which belongs to either of the following groups: *informational*, *acceptance*, *rejection*, *failure* and *redirection*. This classification is based on the meaning and information that each response provides.

The requests can be classified as follows.

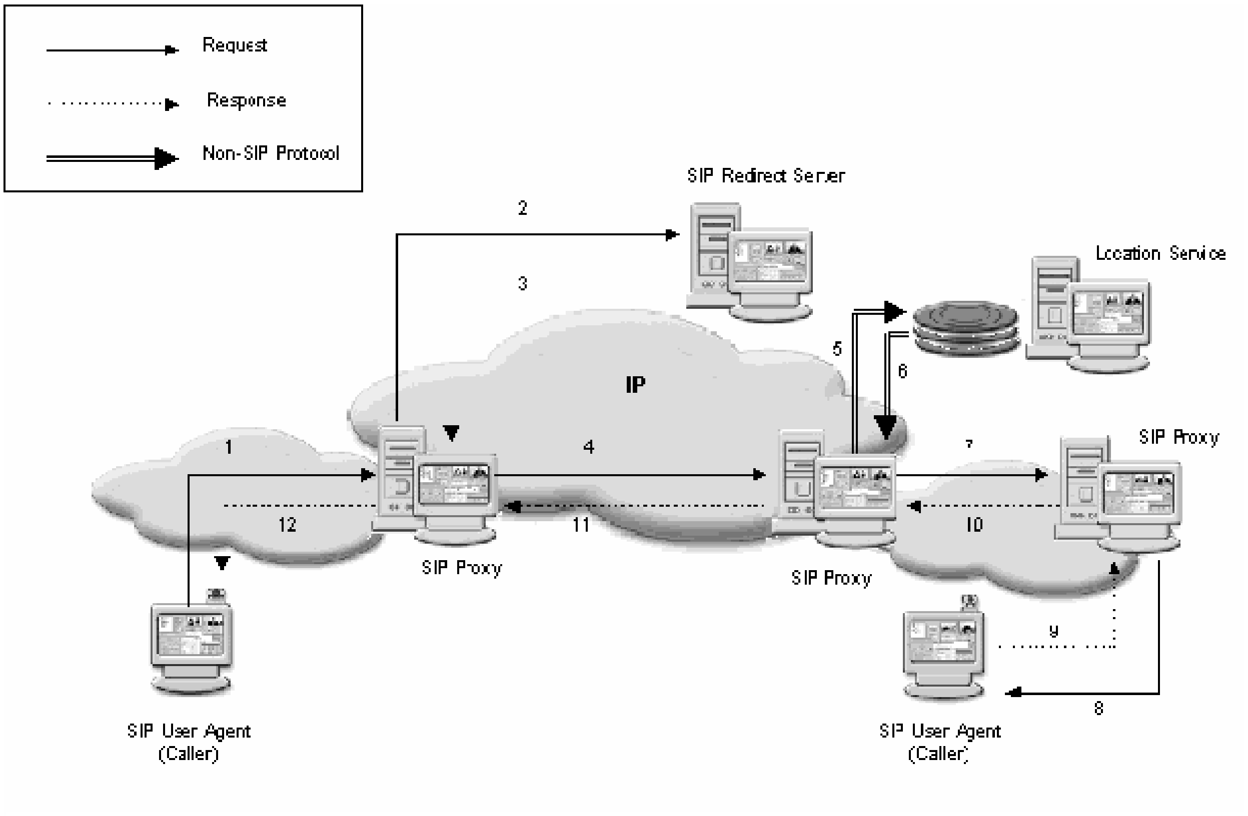

Figure 4: General Architecture for SIP

3.1 Classification of Requests

Each *request* is identified by a method field in the SIP message which defines the purpose of the *request*. The component which receives a *request* responds according to the method field. Therefore, the methods define the meaning of each *request* and the behaviour of the servers vary accordingly. In SIP version 2.0, following request methods are available:

Here we give a brief explanation of each method:

- **Invite :** The *Invite* request is sent to invite a user to participate in a session. The message body contains a description of the session to which the callee is being invited.
- **Ack :** The *Ack* request sent by the caller acknowledges that the caller has received an *Ok* response to an *Invite* request. *Ack* is used only with *Invite* request. If the proxy server between caller and callee receives some other response apart from *Ok*, it does not send an *Ack*.
- **Options :** The *Options* request asks a server about its capabilities. A server that believes it can contact the user where the user is logged in and has been recently active, may respond to this request with its status showing the user's whereabouts. A user agent receiving an *Options* request may return a status which reflects how it would have responded to an invitation, e.g., busy. Proxy and redirect servers simply forward the request without indicating their capabilities.
- **Bye :** The *user agent* uses *Bye* to indicate that it wishes to release the call. A *Bye* request is forwarded like an *Invite* request and may be issued by either the caller or the callee. A party should issue a BYE request before releasing a call (hanging up). A party receiving a BYE request must cease transmitting media streams specifically directed to the party issuing the BYE request.
- **Cancel :** The *Cancel* request cancels a pending request, but does not affect a completed request. (A request is considered to be completed if the server has returned a final status response.)
- **Register :** A user agent uses *Register* method to register the address listed in the To header field with the SIP server. A user agent may register with a local server on startup by sending a REGISTER request to all SIP servers (multicast address sip.mcast.net (224.0.1.75)).
- Info: mid-call information (ISUP, DTMF)
- Comet: precondition met

- Prack: provisional acknowledgement
- Subscribe: subscribe to event
- Notify: notify subscribers

A successful SIP session in establishing a connection consists of two requests, an *Invite* followed by an *Ack* request. The request *Invite* asks the callee to join a particular conference or to establish a two party conversation. After the callee has agreed to participate in the call, the caller confirms that it has received the response from callee by sending a request including an ACK method. If the caller no longer wants to participate in the call, it sends a BYE method in a request instead of an ACK. Note that the *requests* that are not supported by a proxy or redirect server are treated by that server as if they were an OPTIONS request and forwarded accordingly. Requests that are not supported by a user agent server or registrar cause a 501(Not Implemented, see Appendix for details) response to be returned.

3.2 Classification of Responses

The possible responses that can be received are classified by classes of numbers as follows:

- **Class 1xx - Informational :** Informational responses indicate that the server or proxy contacted is performing some other action and it is unable to send a definitive response. The client should wait for a further response from the server and the server should send such a response without further prompting. A server should send a 1xx response if it expects to take more than 200ms to generate a final response. A server may issue zero or more 1xx responses, with no restrictions on their ordering or uniqueness. 1xx responses do not cause the client to send an ACK. Servers are free to transmit informational responses and clients can inquire about the current state of call processing by the re-sending the request. Examples of 1xx class are: 100 (Trying) , 180 (Ringing), etc.

- **Class 2xx :** 2xx responses mean that the request was successful and must terminate a search. The example for this class is 200 (it Ok).

- **Class 3xx - Redirection :** 3xx responses give information about the user's new location or about alternative services that might be able to satisfy the call. They should terminate an existing search and may cause the initiator to begin a new search, if appropriate. To avoid forwarding loops, a user agent client or proxy must check whether the address returned by a redirect server equals an address tried earlier. Examples for this class are: 302 (Moved Temporarily).

- **Class 4xx Request Failure :** 4xx responses are definite failure responses from a particular server. The client should not retry the same request without modification (e.g., adding appropriate authorization). However, the same request to a different server might be successful. Example : 400 (bad Request).
- **Class 5xx Server Failure :** 5xx responses are failure responses. They are generated when a server itself has failed. They are not definitive failures, and must not terminate a search, if other possible locations remain untried.
- **Class 6xx Global Failures :** 6xx responses indicate that a server has definitive information about a particular user, not just the particular instance indicated in the Request URL. All further searches for this user are doomed to failure and pending searches should be terminated.

All responses of these classes are described in detail in Appendix.

4 Some Execution Sequences for Invite and Cancel in SIP

Fig. 5 gives the order of the messages used to initiate a session between two participants using a direct communication without a proxy server as a mediator.

- Fig. 5 shows the interaction between two user agents without using an intermediate server like proxy server or redirect server. It also shows the order and the type of messages and the parties involved in each message. Here, it is assumed that the caller already knows a valid IP address for the callee and contacts directly. A brief explanation for the diagram follows:

  The user agent client (UAC) which is the user in the left side of the figure sends an *Invite* request to the user agent server (UAS). The UAS responds with a 200 it Ok response and the UAC sends an acknowledgement *Ack* back to the UAS. The session starts and the media is transferred for a while. When the caller wants to terminate the session, it sends a *Bye* request to the UAS and the session terminates when UAS responds with a 200 it Ok response.
- Figure 6 corresponds to the message sequence, where a client wants to start a conference with two other participants. The transmission starts when the caller sends an *Invite* request for Alice to the proxy server. The proxy server forwards this message to the Internet location (IP address) where Alice is. Alice responds with a 100 trying to the *Invite* request and then responds with a 486 busy. The proxy server Acknowledges this response and sends an invite request to User agent server where Bob is located. Bob responds with a 180

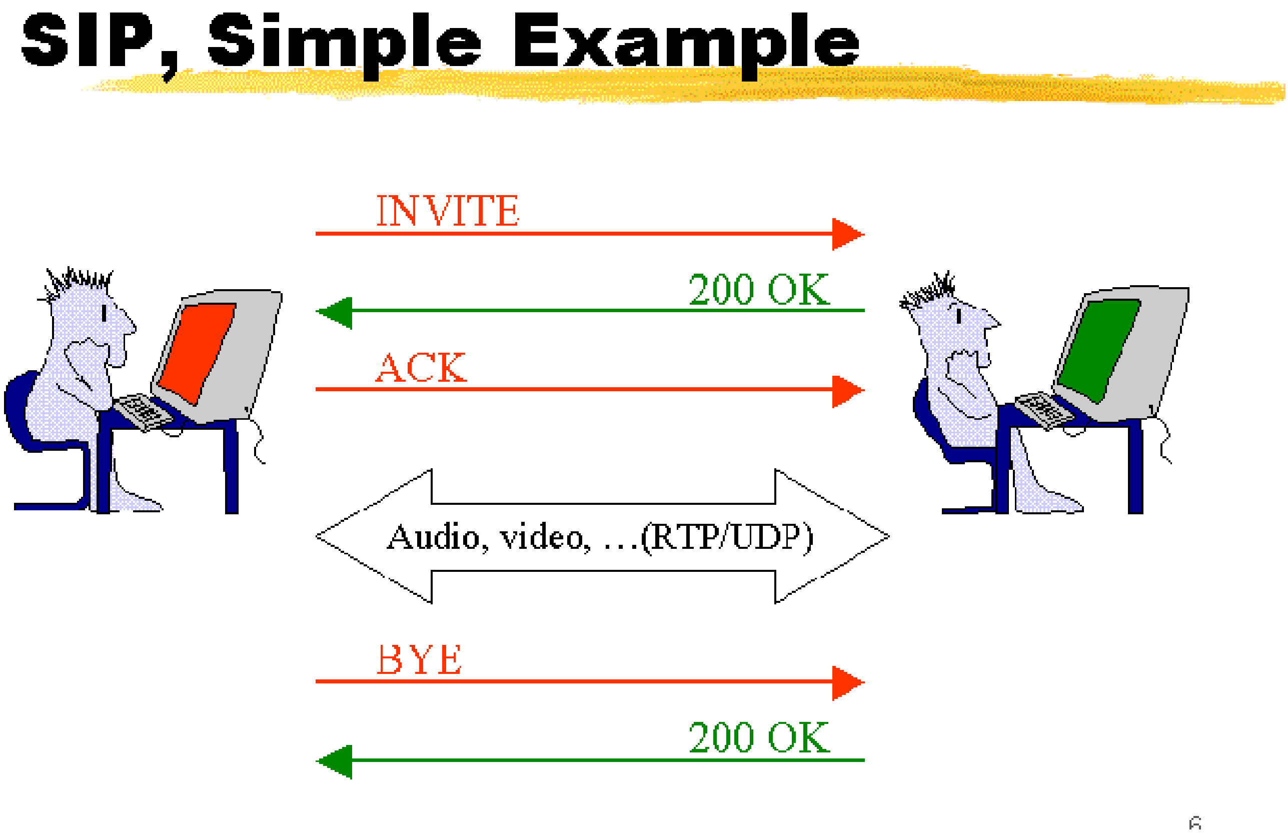

Figure 5: Direct Communication

ringing response. The proxy server forwards this response back to the caller UAC. After Bob answers to the ringing, the UAS for Bob generates a 200 it Ok response and sends it to the proxy server. The proxy server forwards this response also back to the caller or user agent client. The caller then generates an acknowledgement and sends it to the proxy server. The proxy server forwards this ACK to bob and the session starts between the caller and Bob.

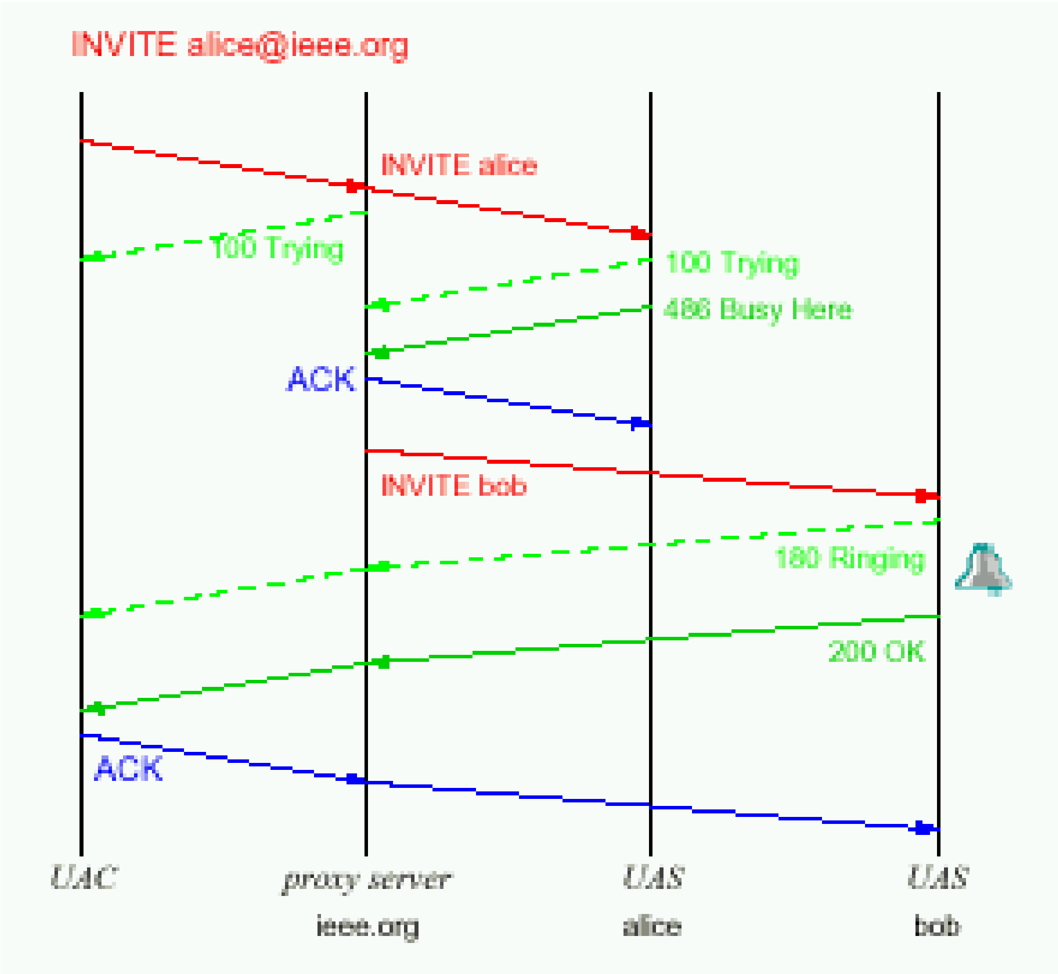

Figure 6: Communication through Proxy server

- Fig. 7 illustrates the mobility feature in SIP as follows:

  A user might have more than one SIP addresses registered for the same permanent address in the *Registrar*. The message sequence in Fig. 7 shows how the server finds the callee in one of the two addresses found and cancels the other *Invite* request. The transmission starts by an Invite request for Bob sent from the caller to the proxy server (the proxy server receives two addresses from the registrar for Bob. The proxy server forwards th*Invite* request to both addresses and sends a (100) trying response back to the caller. The user agent servers which have recieved the *Invites*, send 180 ringing response to the proxy server and the proxy server forwards these responses to the caller. Bob answers to the ringing at the address at home and the UAS generates a 200 *Ok* response to the proxy server. The proxy server forwards the 200 *Ok* response to the caller and then cancels the *Invite* which has been sent to the UAS corresponding to the work address for Bob by sending a Cancel request to that location. The UAS for the address work for Bob, sends a 200 it Ok back to the proxy server for the *Cancel* request it has recieved and then it generates and sends a 487 cancelled response to the proxy server. The proxy server acknowledges this response by an *Ack* request to the work address and finally the caller acknowleges the UAS at home address of Bob by an *Ack* request to establish the session and the media transmission starts between Alice and Bob at his home address.

5 Behaviour of the clients and servers and the state diagrams

5.1 State diagram of the Client for Invite Request

5.1.1 Nomenclature of the automata

A state diagram is a graph where the nodes are representing the states of a system and the edges between the nodes represent transitions. The edges are labelled to define which action is performed during the transition. There are three types of transitions :

- *message!componentname* : The content identified by *message* is sent to the component identified by *componentname.*
- *message?componentname*: The content identified by *message* is received from the component identified by *componentname.*
- T: Unobservable action or internal action. It is used to define actions that are not observable by the environment.

We represent a state diagram for the User Agent Client in Figure. 8 with a high level of abstraction. A client is the party (process) that starts a session by sending an invitation to another party (server process) which in return interacts with the client by sending responses (a separate transaction). The state diagram for User agent

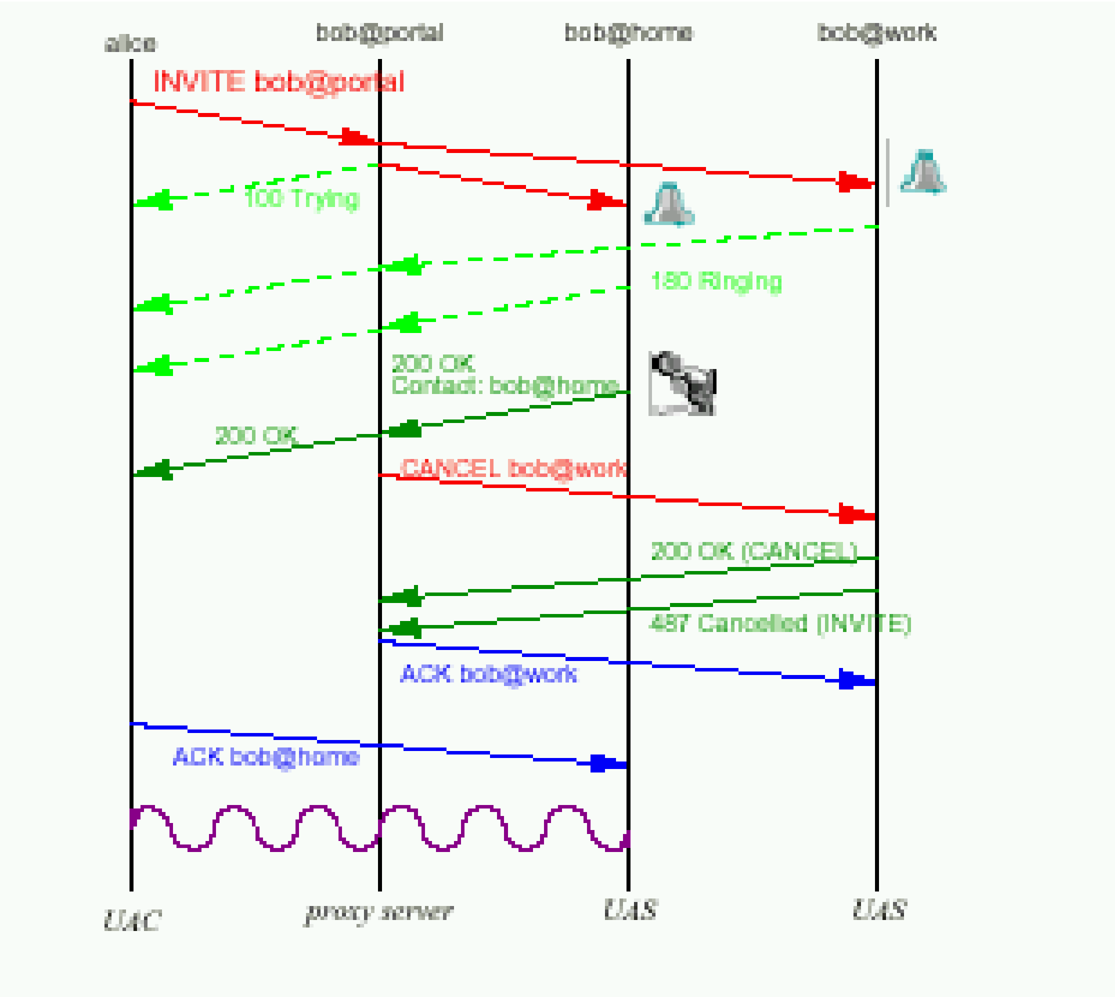

Figure 7: Parallel search with cancel

client has a starting state in which the interaction starts, which is called Initial, and 4 final states in which the process terminates. These states are Dying, Closed1, Closed2 and Closed3.

The session starts from the Initial state by generating an *Invite* request to a defined user. The state of the client therefore changes to Calling state. There are several possibilities in this state. If there is no response from the server, the client generates another *Invite* request and maintains its state. (The time for which the client waits to generate a new *Invite* request is doubled each time.) If the received response belongs to the class 3xx Redirection, then a new *Invite* request is generated to the new address received and the client preserves its state.

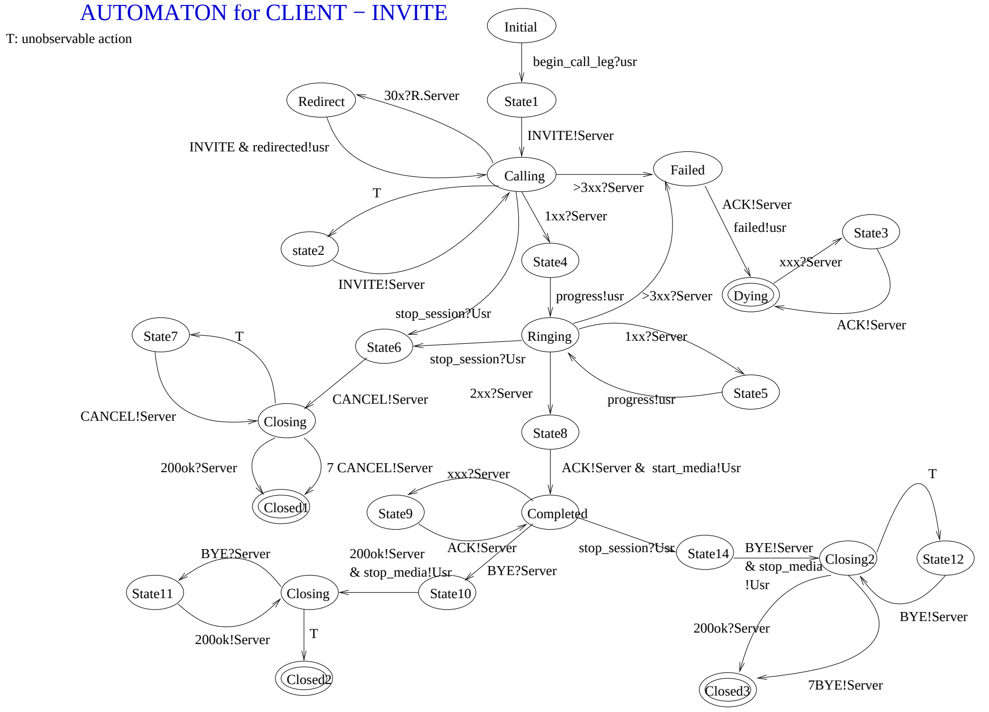

Figure 8: Automaton for Client sending Invite Request

Request is terminated: If the response generated by the server is a stop-session, then the client generates a *Cancel* request and its state changes to Closing.

While in Calling state, if the client receives a stop-session from the server, it reaches Closing state. If it receives a response which belongs to one of the classes 4xx, 5xx or 6xx), then the client generates an *Ack* and changes its state from Calling to Dying.

In the Dying state, the client process generates ACK for any received response. If there is no response, it waits for some time and terminates.

In the Closing state, the client generates a *Cancel* request.(It repeats this behavour until it receives a response). If there is no response after sending 7 *Cancels* request (for TCP) or 11 *Cancels* (for UDP), the client process terminates. If there is a 200 *Ok* response received, the client generates no further request and terminates.

The Request is successful : In the Calling state, the received response might belong to the Class 1xx, Informational. In this case the client changes its state to Ringing.

While the client is in the Ringing state, it might receive a response greater than 400. As in the Calling state, the client generates an ACK request and changes its state to the Dying state. In the Ringing state, the client continues to communicate with the server (progress) while it receives 1xx class responses. If the received response is a stop-session, then in a similar manner to the Calling state, the state is changed from Ringing to the Closing. If the client receives a class 2xx response from the server while Ringing, then it generates an ACK request and the media starts. The state also changes to Completed.

In the Completed state, the client generates *Ack* in return of any received response and maintains its state. If it receives a *Bye* request from the server, the client generates a 200 *Ok* response and terminates the media and changes its state to Closing1. If it receives a stop session response from the server, it generates a *Bye* request and stops the media and changes its state to the Closing 2.

The client in Closing 1 state, sends a *Bye* request in response to any 200 *Ok* and terminates. If there is no response received, it terminates after a while.

And finally in state Closing 2, the client generates a BYE request. If it receives no response for 7 Byes, it terminates. If the client receives a 200 *Ok* response for its own BYE request, it terminates without further generating messages.

5.2 State diagram of the user agent server for Invite Request

The automaton for the server in Figure. 9 shows the transitions and states which server reaches when recieving an Invite request. The state diagram has one initial state called Initial and one final state called Dying. The explanation of the diagram is as following: When the server receives an Invite request from the client, it responds with a class 1xx responses and the state changes to Call-proceeding. While in Call-proceeding state, the server remains in its state in return to any other Invite request it receives. Here the server generates no response. If the server recieves a *Cancel* request from the client, it generates a 200 *Ok* response and change its state to Dying. While in Call-proceeding, if the server declines the *Invite* for a session, it generates a response greater or equal to 300 and changes its state to Failure. It declines *Invite* in the following cases :

1. If an UAS which is not acting as a gateway, receives an Invite with a Request-URL which does not belong to its configured addresses, it declines *Invite* and returns 404.

2. Also, when the server receives an Invite before it sent its final response to the previous invite with a lower sequence number, it sends a 400 (Bad Request) response.

3. if User agent A sends an *Invite* to B and receives an Invite from B before it receives a response for its own *Invite*, it returns a 500 (Internal Server Error).

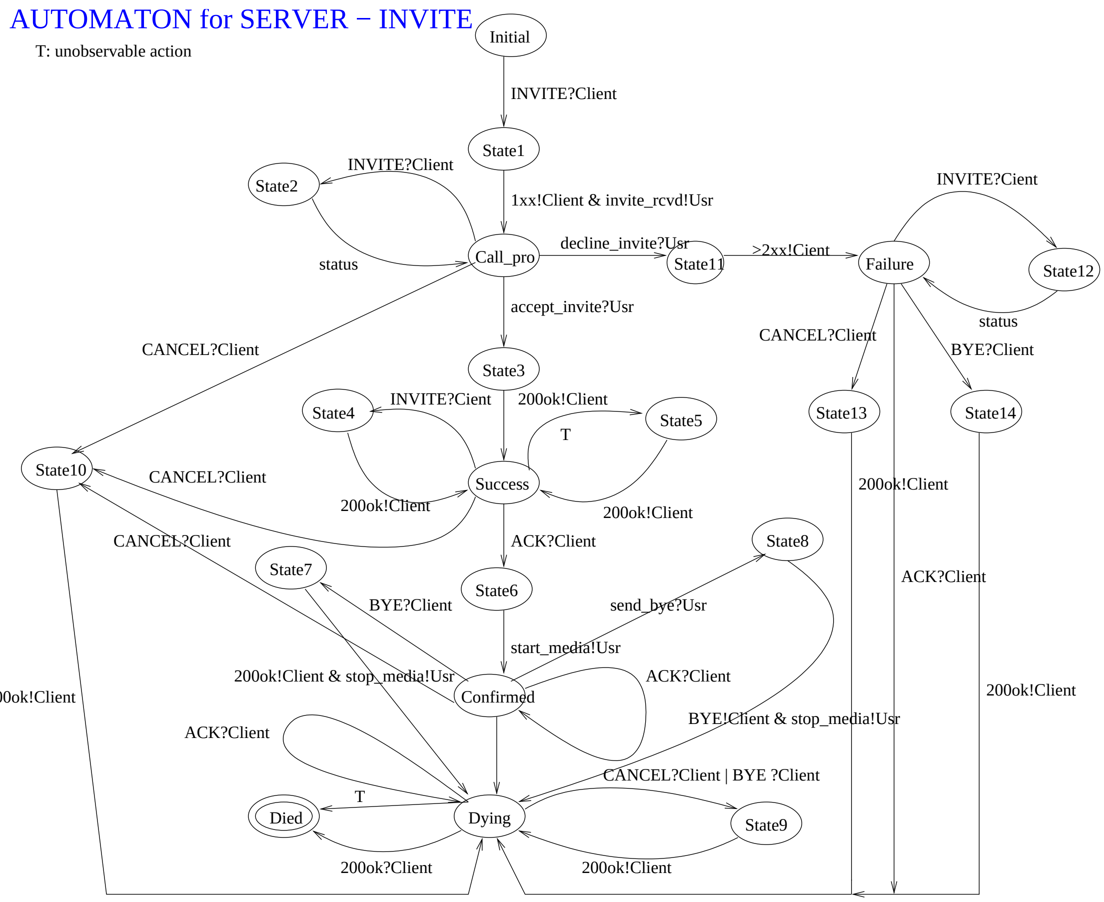

Figure 9: Automaton for Server receiving Invite Request

While in Failure state, the server remains in its status for any received Invite request without generating any response. If an Acknowledgment is received, the state changes from Failure to Dying with no response generated. If a *Bye* request is received, the server generates a 200 *Ok* response and changes its state to Dying. If a *Cancel* request is received, the server responds with a 200 *Ok* response and changes its state to Dying.

But if the server accepts the Invitation from client, it generates a 200 *Ok* and changes its state from Call-proceeding to Success.

While in Success state, the server responds to every received *Invite* request with a 200 *Ok* response. If a *Cancel* request is received in this state, the server responds with a 200 it Ok and changes its state to Dying. If there is no request received at all, the server keeps generating 200 *Ok* responses to the client.

While in Success state, if the server receives an Acknowledgment from the client, it starts the media and changes to the Confirmed state.

While in Confirmed state, if the server receives a Cancel request, it generates a 200 *Ok* response and its state changes to Dying. The server generates no response and remains in Confirmed state for Ackrequests. In Confirmed state, if the server receives a Bye request, it responds with a 200 *Ok* and stops the media. If the server wants to stop the media with Bye, it generates a Bye request and stops the media. In both cases, the state changes from Confirmed to Dying.

And finally while in Dying state, for any received Cancel or Bye request, the server responds with a 200 *Ok* and remains in its state. If an Ack request is received, the server generates no response and remains in its state. If a 200 *Ok* response or no signal at all is received, the server terminates after a while for the later case and immediately for the first case.

5.3 State diagram of the Redirect Server for Invite Request

The state diagram for the Redirect Server in Figure. 10 explains its behaviour for an Invite request. In this state diagram, the transitions are labelled with a single action. In each transition, either a request is received from the caller or a response is sent to the caller. There is an initial state called Init and a final state called Complete. In Init state, an *Invite* is received from the caller. The state of the redirect server is then changed from Init to State1. The redirect server responds to the caller by sending a class 1xx response and changes its state from State1 to Location-query.

While in Location-query, if another *Invite* is recieved from the caller, the state is changed to State2 and the redirect server responds with a class 1xx response and changes its state from State2 back to Location-query. In Location-query state, if a *Cancel* request is received from the caller, the state is changed to State3 and

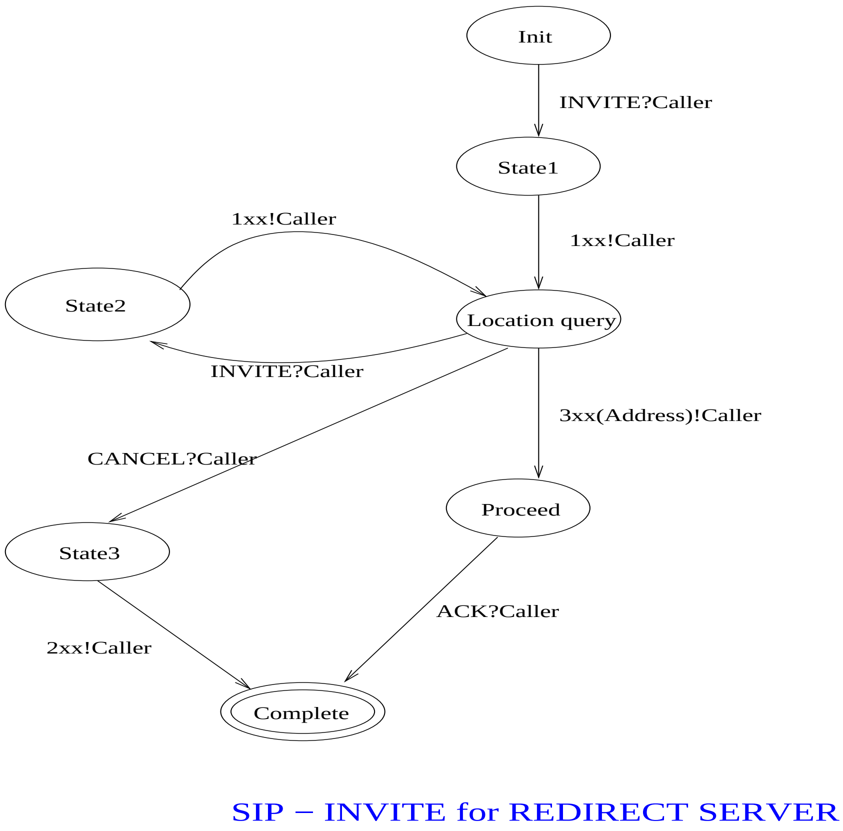

Figure 10: Automaton for Redirect Server receiving Invite Request

the redirect server responds with a 200 *Ok* to the caller and changes its state to Complete. Otherwise, if there is no *Cancel* received, the redirect server responds with a class 3xx redirect response to the caller and sends the correct location along with the response. At this point, the state changes from Location-query to Proceed. It is important to note that while in Location-query, the redirect server contacts the *Registrar* to find the location of the callee.

The redirect server in the Proceed state waits for an acknowledgement from the caller. When it receives an *Ack* request from the caller, it changes its state to Complete.

5.4 State diagram of the Proxy Server for Invite Request

The state diagram of the proxy server for Invite request in Figure. 11demonstrates the behaviour of the Proxy server's role in a SIP interaction. The diagram has an initial state called Init and a final state called Completed. In Init state, an Invite request is received from the caller. The state of the proxy server changes from Init to State1. The proxy server sends a class 1xx response to the caller and changes its state from State1 to Location-query, as it was for the case of the redirect server.

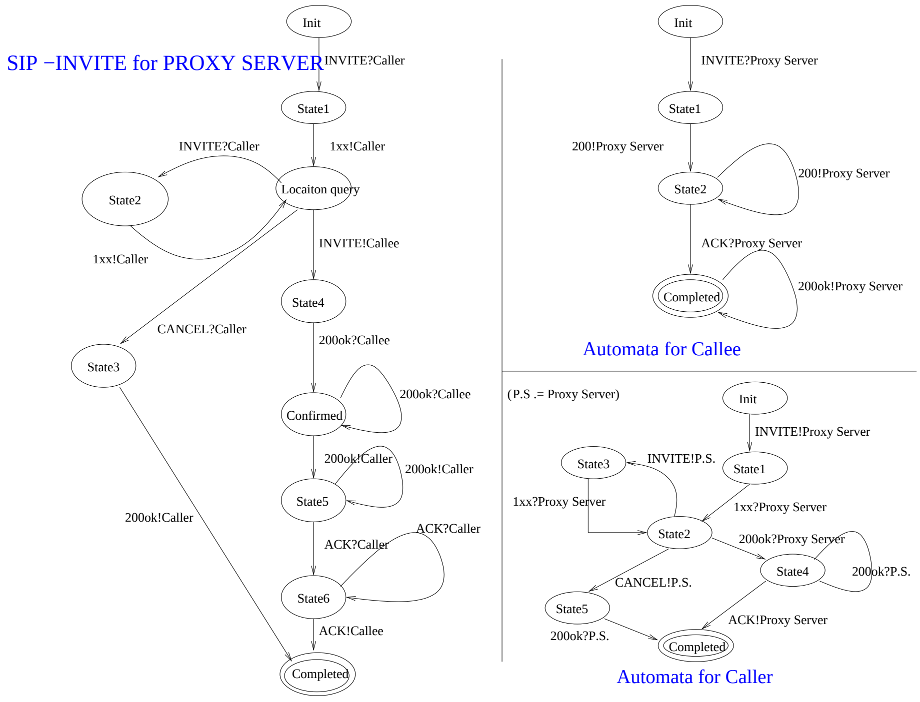

Figure 11: Automaton for Proxy Server receiving Invite Request

The proxy server fetches the address of the callee from the *Registrar* in this state. If another Invite is received from the caller in this state, the proxy server changes its state to State2 and responds with a class 1xx response and changes its state back to Location query.

If a *Cancel* request is received from the caller while in Location-query, the state is changed to State3. The proxy server then responds to the caller with a 200 *Ok* response and changes its state from State3 to Completed.

While in Location-query, the proxy server forwards the *Invite* message it has previously received to the callee and changes its state to State4. If a 200 *Ok* response is received from the callee, the proxy server changes its state from State4 to Confirmed.

While in Confirmed state, the proxy server remains in this state for any other 200 *Ok* responses that it receives from the callee. In the Confirmed state, the proxy server then forwards the 200 *Ok* message received from callee to the caller and changes its state to State5.

In State5, the proxy server waits for an *Ack* request to be recieved from caller. When this request is received, the proxy server changes its state once more from State5 to State6.

In State6, the proxy server remains in its state while receiving any other *Ack* request from the caller. The proxy server then forwards the received *Ack* request to the callee and changes its state from State6 to Completed. We have also included here the automata for the caller and callee separately for this transaction with the Proxy server.

6 Further Work

- We need to define different properties for different behaviours of SIP.
- We also need to refine the automata by including the infinite state aspects of SIP.

A Detailed Description of the different classes of Responses

A.1 Class 1xx

100 Trying : Some unspecified action is being taken place on behalf of this call (e.g., a database is being consulted), but the user has not been located.

180 Ringing : The called user agent has located a possible location where the user has registered recently and is trying to alert the user.

181 Call is Being Forwarded : A proxy server may use this status code to indicate that the call is being forwarded to a different set of destinations.

182 Queued : The called party is temporarily unavailable, but the callee has decided to queue the call rather than reject it. When the callee becomes available, it will return the appropriate final status response. The reason phrase may give further details about the status of the call, e.g., 5 calls queued; expected waiting time is 15 minutes. The server may issue several 182 responses to update the caller about the status of the queued call.

A.2 Class 2xx

200 Ok : The request has succeeded. The information returned with the response depends on the method used in the request, for example: BYE, The call has been terminated. The message body is empty. CANCEL: The search has been cancelled. The message body is empty. The 200 ok response includes the following with respect to the different requests it is generated for. For INVITE, it indicates that the callee has agreed to participate and the message body indicates the callee's capabilities. For OPTIONS, it indicates that the callee has agreed to share its capabilities included in the message body. For REGISTER, it indicates that the registration has succeeded. The client treats the message body according to its content type.

A.3 Class 3xx

300 Multiple choices : The address in the request is resolved to several choices, each with its own specific location and the user can select a preferred communication end point and redirect its request to that location.

301 Moved Permanently : The user can no longer be found at the address in the requested URL and the requesting client should retry at the new address given by the Contact header field. The caller should update any local directories, address books and user location caches with this new value and redirect future requests to the addresses listed.

302 Moved Temporarily : The requesting client should retry the request at the new address(es) given by the Contact header field. The duration of the redirection can be indicated through an Expiry header field. If there is no explicit expiration time, the address is only valid for this call and must not be cached for future calls.

305 Use Proxy : The requested resource must be accessed through the proxy given by the Contact field. The Contact field gives the URL of the proxy. The recipient is expected to repeat this single request via the proxy. 305 responses must only be generated by the user agent servers.

A.4 Class 4xx

400 Bad Request : The request could not be understood due to malformed syntax.

401 Unauthorized : The request requires user authorization.

402 Payment Required : Reserved for future use.

403 Forbidden : The server understood the request, but is refusing to fulfil it. Authorization will not help and the requests should not be repeated.

A.5 Class 5xx

500 Server Internal Error : The server encountered an unexpected condition that prevented it from fulfilling the request. The client may display the specific error condition and may retry the request after several seconds.

501 Not Implemented : The server does not support the functionality required to fulfil the request. This is the appropriate response when server does not recognize the request method and is not capable of supporting it for any user.

502 Bad Gateway : The server, while acting as a gateway or proxy, received an invalid response from the downstream server it accessed in attempting to fulfil the request.

503 Service Unavailable : The server is currently unable to handle the requets due to a temporary overloading or maintenance of the server. The implication is that this is a temporary condition which will be alleviated after some delay. If known, the length of the delay may be indicated in a retry-after header. If no retry-after header is given, the client must handle the response as it would for 500 response.

504 Gateway Time-out : The server, while acting as a gateway, did not receive a timely response from the server (e.g., a location server) it accessed in attempting to complete the request.

A.6 Class 6xx

600 Busy Everywhere : The callee's end system was contacted successfully but the callee is busy and does not wish to take the call at this time. The response may indicate a better time to call in the retry-after header. If the callee does not wish to reveal the reason for declining the call, the callee uses status code 603 (Decline) instead. This status response is returned only if the client knows that no other end point (such as a voice mail system) will answer the request. Otherwise, 486 (busy here) should be returned.

603 Decline : The callee's machine was successfully contacted but the user explicitly does not wish to or can not participate. The response may indicate a better time to call in the retry-after header.

604 Does not exist anywhere : The server has authoritative information that the user indicated in the To request field does not exist anywhere. Searching for the user elsewhere, will not yield any results.

606 Not Acceptable : The user agent was contacted successfully, but some aspects of the session description such as the request media, bandwidth or addressing style were not acceptable.